**Report Document: Huffman Compression and Decompression Program**

**1. Project Overview**

This project implements Huffman coding for file compression and decompression. The program uses Huffman trees to generate variable-length binary codes based on character frequencies in the input file. The compressed files have a .abiz extension, and decompressed files restore the original content in a new output file.

**2. Files and Structure**

* **Header**: Contains the metadata needed for decompression, including character frequency counts, Huffman codes for each character, and padding information.
* **Input File**: The original file to be compressed or the .abiz file to be decompressed.
* **Output File**: The compressed file with the .abiz extension or the decompressed file as output{filename}.

**3. Namespaces and Modules**

* **Namespace: Huffman**:
  + Houses all Huffman coding-related utilities.
  + Contains sub-namespaces for compression and decompression utilities.
* **Sub-namespaces**:
  + **Utility**: Common functions for both compression and decompression, including functions to get file size and traverse the Huffman tree.
  + **CompressUtility**: Functions for generating character frequency, creating the Huffman tree, generating Huffman codes, and compressing the file.
  + **DecompressUtility**: Functions for reading and decoding the Huffman header, reconstructing the Huffman tree, and decompressing the .abiz file.

**4. Detailed Breakdown of Code Components**

**A. Utility Functions**

1. **File Size Calculation**: Calculates the size of the input file using file pointer operations.
2. **Tree Traversal**: Inorder traversal function for printing Huffman codes of each character.

**B. Compression Functions**

1. **Character Frequency Parsing**:
   * parse\_file: Reads the file byte-by-byte to count character frequencies, then stores these in a map.
2. **Huffman Tree Generation**:
   * generate\_huffman\_tree: Creates a binary tree where each node represents a character and its frequency.
   * combine: Combines nodes based on frequency to form the Huffman tree structure.
3. **Header Generation**:
   * generate\_header: Creates a compressed header that includes unique character count, Huffman codes, and padding data.
4. **File Compression**:
   * compress: Compresses the file by encoding each character with its Huffman code, managing padding as necessary.

**C. Decompression Functions**

1. **Header Decoding**:
   * decode\_header: Reads and reconstructs the Huffman tree from the compressed file's header.
2. **Huffman Tree Reconstruction**:
   * generate\_huffman\_tree: Rebuilds the tree structure to decode characters based on bit patterns.
3. **File Decompression**:
   * decompress: Reads the .abiz file, decodes it using the Huffman tree, and writes the decompressed data to an output file.

**5. Compression and Decompression Flow**

1. **Compression**:
   * Reads file character-by-character.
   * Generates Huffman codes based on character frequency.
   * Stores Huffman codes in a header and compresses the main content using these codes.
   * Adds padding to ensure byte alignment.
2. **Decompression**:
   * Reads header to retrieve Huffman codes.
   * Rebuilds the Huffman tree and decodes each byte according to the stored codes.
   * Outputs the decompressed content to a file.

**6. Execution Details**

* **Main Program Flow**:
  + Prompts the user to specify a file and mode (compression or decompression).
  + Measures time taken for compression or decompression.
  + Outputs file sizes and compression ratios.
* **Example Usage**:
  + **Compression**:

bash

Copy code

Enter the filename: example.txt

Enter the mode (-c for compress, -dc for decompress): -c

* + **Decompression**:

bash

Copy code

Enter the filename: example.txt.abiz

Enter the mode (-c for compress, -dc for decompress): -dc

**7. Efficiency and Memory Considerations**

* The program is optimized for memory and processing efficiency by:
  + Using vectors and maps for frequency storage.
  + Managing padding to align compressed data to bytes.
  + Only storing nodes for unique characters in the tree.
  + Employing sorting techniques for node management in the Huffman tree.
* **Compression Efficiency**: Files with a higher occurrence of a small subset of characters will compress more effectively, as Huffman codes reduce the data size significantly.

**8. Code Performance**

* **File Size and Speed**: Depending on the file's original size and character frequency distribution, compression speed varies but is generally efficient for moderately sized text files.
* **Execution Time**: Compression and decompression times are displayed after each operation for performance assessment.

**9. Known Limitations**

* The program relies on sufficient memory for handling the Huffman tree, which may be limited for extremely large files or files with numerous unique characters.
* Compressed files with highly skewed Huffman trees might lead to increased header sizes, impacting compression ratios.